



(19) Europäisches Patentamt  
European Patent Office  
Office européen des brevets



(11) Publication number: 0 674 277 A2

(12)

## EUROPEAN PATENT APPLICATION

(21) Application number: 95301898.3

(51) Int. Cl.<sup>6</sup>: G06F 17/21

(22) Date of filing: 22.03.95

(30) Priority: 23.03.94 US 216729

(43) Date of publication of application:  
27.09.95 Bulletin 95/39

(84) Designated Contracting States:  
CH DE FR GB IT LI

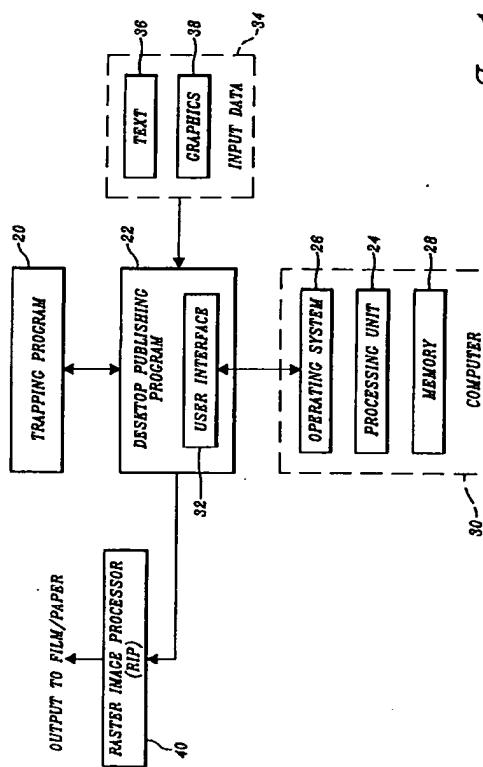
(71) Applicant: ALDUS CORPORATION  
411 First Avenue South  
Seattle, WA 98104-2871 (US)

(72) Inventor: Gartland, Richard A.  
11931 N.E. 168th Street  
Bothell, Washington 98011 (US)

(74) Representative: Spall, Christopher John  
BARKER, BRETELL & DUNCAN  
138 Hagley Road  
Edgbaston Birmingham B16 9PW (GB)

(54) Method of trapping graphical objects in a desktop publishing program.

(57) Disclosed is a method of trapping a publication specified in a PostScript page description language (PDL) or other PDL file by modifying the publication prolog to create traps within an interpreter or RIP.



EP 0 674 277 A2

Cross-Reference to Related Applications

This application is related to U.S. Patent Application Serial No. 08/188,246, filed January 26, 1994, entitled "Applying Traps to a Printed Page Specified in a Page Description Language Format" and assigned to the assignee of the present invention. United States Patent Application Serial No. 08/188,246 is hereby incorporated by reference.

Field of the Invention

The invention generally relates to color trapping and, more particularly, to a method of creating traps for objects in desktop publishing programs.

Background of the Invention

Color printing has traditionally required the integration of many unique and varied talents to see a project through from conception to a printed page. Prior to "desktop publishing," ideas or concepts were typically first drawn by hand and photographed, any text or illustrations added, and the aggregate of pictures and text used to produce a printed page. The traditional process generally required, in addition to design personnel, a paste-up person, typesetting bureau and a lithography department that would produce separations from the photographs.

Desktop publishing has relieved some of the burden of publishers by allowing color production, i.e., drawing and layout, to be integrated electronically using personal computers. Color documents can now be designed, enhanced, color-corrected, and separated on a computer "desktop." The success of desktop publishing is, in large part, the result of standards-based computer programs such as the page description language (PDL) PostScript® from Adobe Systems Incorporated of Mountain View, California. The PostScript language has become the industry standard that serves as an intermediary between applications packages, e.g., desktop publishing programs, and PDL-compatible composite reproduction equipment, such as a desktop printer. The applications packages translate information for a page into PDL for transmission to the PDL-compatible desktop printer. The PDL-compatible printer includes an interpreter that converts the PDL code to low-level instructions that indicate to the printer how to render the text and graphics. Alternatively, the information can be transferred to a file for importation by another applications program or for use by a service bureau. The process of interpreting and rendering PDL is often performed within a raster image processor or "RIP." For further information on the PostScript programming language, please see *PostScript Language Reference Manual*, 2nd Ed., by Adobe Systems, Inc., published by Addison Wesley, which is hereby incorporated by refer-

ence.

A desktop printer uses toner to produce the colors comprising a color publication. Because composite printing is generally efficient only for small quantity jobs, larger quantities of the same publication are generally reproduced on a commercial printing press using ink. Methods of printing color publications using a commercial printer press include process-color printing, spot-color printing, or a combination of the two. Process-color printing separates the original image into its cyan (C), magenta (M), yellow (Y), and black (K) components to recreate the original shadings of color in the publication. This is accomplished by printing dots of the process-color inks in different combinations in close proximity to simulate a variety of colors on a printed page. Spot-color printing involves printing one or more specific colors (or inks) that have been specified according to a color matching system. One popular color matching system is the PANTONE™ MATCHING SYSTEM by Pantone, Inc. Spot-color printing is often used to produce colors that are not easily produced using CMYK inks, such as silver, gold, and fluorescent colors. Spot-color printing is also used in lieu of printing the four process colors, such as when only a couple of inks are required for a particular publication.

Before a color publication can be reproduced on a commercial printing press, each page containing composite art must be separated into its component colors by printing a film separation for each ink (cyan, magenta, yellow, and black, if process colors are to be printed) and any spot colors. Thus, process-color printing always requires four film separations. Spot-color printing requires a separation for each color being printed. A commercial printer uses these film separations to create the printing plates used on the press. For instance, if one specifies the four process colors and a single spot color in a publication, there will be five separations, and hence printing plates, for each page. A separate component ink is added by each plate as the pages in the publication pass through the press. For a more detailed explanation of the commercial printing process, please see the *Commercial Printing Guide* from PageMaker®, Version 5.0. PageMaker® is a desktop publishing program produced by Aldus Corporation, the assignee of the present application. The *PageMaker 5.0 User Manual* and *Commercial Printing Guide* are hereby incorporated by reference. For additional information on desktop publishing generally, see *Desktop Publishing in Color* by Michael Kieran, published by Bantam Books (1991), which is also incorporated by reference.

High-quality printing, such as that required by the publishing industry, poses many difficult problems in controlling the separations. For example, color printing is compromised if paper is not properly aligned as it makes multiple passes through the plates of a prin-

ter. This problem is typically referred to as misregistration. One common solution to the registration problem is to perform a technique known as trapping. Trapping refers to expanding or "spreading" regions of a particular color beyond its normal boundaries, and contracting or "choking" a color region so that a small overlap exists between graphical objects where misregistration may occur. Trapping techniques have traditionally been performed manually. Although tedious, in the past, manual trapping techniques have been used in applications such as magazine publishing, where the time and labor required to create individual traps for each printed page are economically justified.

In recent years, computer systems that perform choking and spreading electronically have come into widespread use. A typical approach has been to: (1) start with a PDL file such as a PostScript file; (2) convert the vector graphics and/or text within the PDL file into a raster (bit mapped) image through a RIP; and (3) trap the raster image using pixel data comprising the image. The third step usually requires a creation of a separate frame buffer for each of the process colors. Each frame buffer is then choked and spread on a pixel-by-pixel basis, and the result used to control the printing of its respective color. As will be appreciated, this approach is very memory intensive.

A more recent approach to electronic trapping is set forth in U.S. Patent No. 5,295,236 ("236 patent"), titled "Applying Traps to a Printed Page Specified in a Page Description Language Format" and assigned to the assignee of the present invention. The '236 patent discloses a method and apparatus for electronically trapping a printed color page in a desktop publishing, layout, graphics, or similar applications program. The method translates PDL instructions comprising the printed page into a format suitable for detection and analysis of edges between color regions in the printed page; creates, according to a set of trapping rules, a set of traps between the color edges; and produces a trap output file that includes the traps expressed in the PDL format. Such a method is referred to as a "post-processing approach." An advantage to the method described in the '236 patent is that virtually any printed page that is expressed in a PDL format may be trapped regardless of the application that originally created the graphics, i.e., the originating program. However, this capability is at the expense of software complexity. In particular, the PDL file must be interpreted before traps are created.

Another approach to electronic trapping is to have the originating program also create traps for the graphics. As an example, in drawing programs such as Aldus Freehand®, a user is allowed to add outlines around objects to accommodate trapping. Trapping approaches at the originating program level are beneficial because additional trapping costs may be eliminated, i.e., it is not necessary to utilize a post-

processing program or traditional trapping techniques. However, the capability to perform trapping in originating programs is relatively new, and has typically required extensive reworking of the computer code comprising the programs. This, in turn, requires a new release of the software, and involves associated costs with such a release, including extensive debugging. Further, trapping capabilities in originating programs have typically been very limited. One shortcoming is apparent in situations where the background and/or foreground is formed by a number of differently colored objects. This situation arises, for example, where individual text characters overlap more than one object. Typical originating programs with trapping capabilities tend to handle objects such as text or boxes in a relatively coarse fashion, such that traps are only able to be applied to an entire text block, or at best an entire character. Thus, the trap results in these instances are less than desirable.

In contrast to the prior art discussed above, the invention alleviates the need to change the computer code of the originating application to provide trapping capabilities and reimplement the PDL interpreter. Hence, a trapping method implemented in accordance with the invention may be created, debugged, and distributed independent of the desktop publishing program. Further, as will be described in detail below, the invention allows very accurate trap placement, thus providing results superior to trapping solutions currently available in originating programs.

#### Summary of the Invention

The invention is a method of electronically trapping a printed page that includes a plurality of objects expressed in a page description language, with the page description language instructions being interpreted by a raster image processor for output to an output device. The method comprises the step of providing instructions to the raster image processor to trap the objects in the page as the page description language instructions comprising the objects are interpreted by the raster image processor.

In accordance with other aspects of the invention, the printed page is in a publication in a desktop publishing program. The publication including a prolog and a script that specify the objects to be printed. The method further includes the step of modifying the publication prolog to provide the trapping instructions to the raster image processor. Further, the script is sent unmodified to the raster image processor.

In accordance with still further aspects of the invention, the printed page is in a publication in a desktop publishing program. The method comprises the steps of: (a) modifying the publication prolog to instruct the raster image processor to (i) create a directory of the color objects in the publication and (ii) trap the color objects in the publication using the shape di-

rectory; and (b) sending the modified publication prolog and script to the raster image processor wherein trap areas are created as the publication is rendered.

#### Brief Description of the Drawings

The above and further advantages of the invention may be better understood by referring to the following description together with the accompanying drawings, wherein:

FIGURE 1 is a block diagram of a trapping program in accordance with the invention that works in conjunction with prior art desktop publishing programs to trap a publication as it is being rasterized by a raster image processor (RIP);

FIGURE 2 is a block diagram illustrating a PDL publication, including a prolog and script, being sent to a RIP by a desktop publishing program in accordance with conventional procedure;

FIGURE 3 is a flow chart illustrating a conventional procedure by which a desktop publishing program sends information to a RIP enabling it to draw a publication;

FIGURE 4 is a flow chart illustrating the internal procedure followed by a conventional RIP to draw an object in a PDL publication;

FIGURE 5 is the block diagram of FIGURE 2 but including revisions that illustrate the sending of additional prolog commands to the RIP to facilitate trapping of the publication in accordance with the invention;

FIGURE 6 is a flow chart of an exemplary routine by which a desktop publishing program sends information to a RIP enabling it to trap a publication in accordance with the invention;

FIGURE 7 is a flow chart of an exemplary routine illustrating how traps are created for objects by the RIP in accordance with the invention;

FIGURE 8 is a flow chart illustrating how objects are trapped within the RIP in accordance with the invention, as a result of the instructions created and sent to the RIP in FIGURE 6;

FIGURE 9 is a flow chart of an exemplary routine in accordance with the invention for revising a trap created for a background object when multiple overlaps interfere with the trap;

FIGURE 10 is a block diagram of a simplified PDL, e.g., PostScript, publication, including a prolog and script written in pseudo code;

FIGURE 11 is a block diagram of the PDL publication illustrated in FIGURE 10, but including modifications to the prolog to provide trapping of the publication during the rendering process at the RIP;

FIGURES 12A-12C illustrate a trapping example involving exemplary objects X and Y to facilitate explanation of the invention;

FIGURES 13A-13C illustrate the trapping exam-

ple of FIGURES 12A-12C but including a third object Z that overlaps the original objects; and FIGURES 14A-14C illustrate the repairing of a previous trap in accordance with the procedure set forth in FIGURE 9.

#### Detailed Description of the Preferred Embodiment

A trapping program 20 in accordance with the invention is illustrated in FIGURE 1. The trapping program 20 works in conjunction with a computer program, such as a desktop publishing program 22, that is suitable for incorporating text, graphics and other aspects of documents to be published. The desktop publishing program 20 may be, for example, the PageMaker® desktop publishing program sold and supported by Aldus Corporation, the assignee of the present invention. It is noted, however, that the benefits of the invention are not limited to use with an application whose primary purpose is to combine text and graphics. Thus, throughout the specification and claims, the term "desktop publishing program" is hereby defined as any computer program that has the ability to manipulate graphical objects, including programs such as presentation, art, and drawing programs.

The trapping and desktop publishing programs 20 and 22 run on a processing unit 24 controlled by an operating system 26. Memory 28 is connected to the processing unit and generally comprises, for example, random access memory (RAM), read only memory (ROM), and magnetic storage media such as a hard drive, floppy disk, or magnetic tape. The processing unit and memory are typically housed within a personal computer 30, including Macintosh™, International Business Machines (IBM™), and IBM-compatible personal computers. When used with IBM and IBM-compatible personal computers, the operating system 26 may incorporate a windowing environment such as Microsoft Windows™.

The desktop publishing program 22 includes a user interface 32 that interacts between the operating system 26 and the internal process application of the desktop publishing program 22. Using the desktop publishing program 22, an author creates the text, images, and graphics comprising a publication. Text, images and graphics are generically referred to as "objects" through the specification and claims. In many cases, the data comprising a publication is also imported from one or more sources including, for example, illustration, image enhancement, word processing, and desktop publishing programs. At block 34, input data including text 36 and graphics 38 that were created within or imported into a publication are shown. After the input data comprising a publication have been entered, the data may be trapped using the trapping program 20. As will be apparent from the following discussion, this is accomplished by sending

instructions to a PostScript (PS) interpreter or raster image processor (RIP) 40, e.g., within a printer or at a service provider, to trap the objects in the publication as the PS instructions comprising the publication are being interpreted. The RIP 40 then outputs the data comprising the trapped publication to film or paper.

Prior to discussing the specific details of how trapping is accomplished by the trapping program 20, background information on the operation of PDL languages, and directed specifically toward the PS language, is provided. While the remainder of this discussion focuses on the PS language, those skilled in the art will appreciate that the ideas presented are applicable to other page description languages as well.

With reference to FIGURE 2, the recommended structure for a PDL file such as a PS publication 50 file is that it contain two basic components: a *prolog* 52 and a *script* 54. The prolog 52 contains routines or procedures, as well as named variables and constants, that will be used throughout the rest of a publication. As is known by those skilled in the art, it is efficient to have a routine to perform a given task that will be repeated multiple times, e.g., drawing a line, box, or oval, and then call the routine with the appropriate parameters. In contrast, a more burdensome approach would be to set forth all of the instructions contained in the routine each time a line, box, or oval is to be drawn. The prolog is written by a PDL programmer, and will precede the first part of every publication, or script, that uses it. The script 54 provides the setup for the publication and describes the specific elements to be produced as the output in terms of procedures and variables defined in the prolog, along with operand data.

When a user of the desktop publishing program enters the print command, the prolog 52 and script 54 are sent to the RIP 40 where the PDL language is interpreted and converted into a bit map format. The bit map is used to draw the objects in the publication on paper or film.

FIGURE 3 is a flow diagram of a prior art routine that illustrates the construction of a publication written in PS language at the interpreter/RIP level. At block 100, the prolog from the publication is sent to the RIP. The next (current) page to be printed is then targeted for "printing" at block 110. Printing in this context refers to sending the PS commands to a RIP, and not necessarily the act of placing ink on paper. At block 112, a test is made to determine whether all objects in the current page have been printed. If all objects in the current page have not been printed, the next object to be considered is selected at block 114. At block 116, the appropriate PS commands are constructed to create the object. The PS commands are then sent to the RIP at block 118, and the routine loops to block 112.

If all the objects in the current page were deter-

mined to be printed at block 112, a test is made at block 120 to determine if all pages in the publication have been printed. If all pages in the publication have not been printed, the routine loops to block 110. Otherwise, the routine is terminated and the RIP is ready to begin converting the PS code comprising the publication into a format acceptable to an output device such as an imagesetter (creating film) or a printer. FIGURE 4 illustrates a simplified, generic version of the process that occurs at the RIP for any given object to be printed. More particularly, at block 122 the parameters for the object are accepted. These were created and sent to the RIP by blocks 116 and 118 of FIGURE 3. The object is then drawn using the parameters, as indicated at block 124. The routine then terminates.

The flow diagram of FIGURE 4 illustrates precisely what PostScript was intended to do: Indicate to an interpreter/RIP objects to be printed and then have the commands carried through. In other words, the function of the RIP has traditionally been to simply draw objects in a publication in accordance with the PDL commands sent to it. In contrast, the invention extends the function of the RIP beyond traditional concepts by instructing the RIP to create traps for objects at the *interpretation/rasterization* level. FIGURE 5 illustrates this concept pictorially. The procedure is similar to FIGURE 2, but includes the following new steps. First, prolog commands 122 that instruct the RIP to trap objects in the publication 50 are created by the trapping program 20 and forwarded to the RIP, along with the traditional prolog commands and the script comprising a publication. More specifically, the prolog 54 is substituted with a trapping prolog 124. It is noted that the script itself does *not* change, and thus this portion of the code within the desktop publishing program 22 need not be modified.

At this point, all control of the publication is at the RIP 40. The RIP interprets the PS instructions in the trap prolog 124, which indicate that a listing, referred to as the "shape directory" 126, is to be created of the objects in the publication as the objects are drawn. Additional PS instructions in the trap prolog 124 indicate that the objects in the publication, including text and graphics, are to be trapped using the shape directory, with the objects and resultant traps being rasterized and sent to an output device by the RIP.

FIGURE 6 illustrates an exemplary routine for implementing the trapping program 20 in conjunction with a desktop publishing program. At block 130, a test is made to determine whether a user wishes to trap a publication in the desktop publishing program. In one embodiment, the user will indicate that a publication is to be trapped prior to invoking the "print" command. If the current publication is to be trapped, at block 132 the preamble normally associated with the publication is replaced with a trap prolog. It will be apparent to those skilled in the art how to facilitate

substitution of the normal prolog with the trap prolog for the desktop publishing program of interest.. For example, in the Macintosh™ version of the PageMaker 5.0 program, the prolog used for a given publication will typically be the one that was most recently opened in the current session. Thus, the trapping program will ensure that the trap prolog is referenced if trapping is desired, and not otherwise. As an alternative to this scheme, the PageMaker 5.0 program will look for a particular file that, if it contains a prolog, will override earlier versions of the prolog. This option may be used in lieu of the one described above. Similarly, other desktop publishing programs have methods of accomplishing the substitution.

After the prologs have been substituted, the trap prolog is sent to the RIP at block 100. If the publication is not to be trapped, the routine skips the prolog replacement step at block 132, and loops to block 100 where the normal prolog is sent to the RIP. Blocks 110-120 have identical functions as those shown and discussed relative to FIGURE 3.

FIGURE 7 illustrates the steps performed by the RIP in printing an object after receiving the trap prolog from the trapping program 20 and the script from the desktop publishing program 22. At block 170, the parameters from the script are accepted for an object to be printed, termed the "current object." At block 172, the object is drawn using these parameters. The object is added to a shape directory at block 174. In a preferred embodiment, only graphical objects are stored in the shape directory, although it may be advantageous to include text in the listing in certain applications. A test is made at block 176 to determine if there are any additional objects in the shape directory. If there are objects in the shape directory, the current object is trapped against those objects, as indicated at block 178. Once the current object has been trapped, or if there were no additional objects in the shape directory, the routine terminates.

FIGURE 8 is a routine suitable for use in FIGURE 7 for trapping the current object being printed in a publication. At block 180, an object from the shape directory is retrieved. The most recently retrieved object from the shape directory is referred to as the "background object" throughout the flow diagram and this discussion. In a preferred embodiment, the objects in the shape directory are retrieved on a first-in, first-out basis, with the bottom-most objects being among the first to be added to the shape directory, and the top-most object being last. Those skilled in the art will appreciate that this bottom-to-top scheme is a direct result of the intricacies of how PostScript handles objects and clipping of objects internally. Other page description languages, or future versions of PostScript, might support a top-to-bottom or hybrid approach, if this is desirable. In this sense, the particular retrieval order is not germane to the invention.

At block 182, a test is made to determine whether

the current object overlaps the background object. As will be appreciated by those skilled in the art, if the current object does not overlap the background object, there is no need to create a trap between these objects. If the current object overlaps the background object, a clipping path is set to the outline of the background object at block 184. The effect of setting the clipping path will be for the RIP to ignore everything on the page except for the background object and any object that overlaps the background object. At block 186, a subroutine Repair\_Trap is called. This routine will remove parts of unwanted traps that were added but are not ideal solutions because of multiple overlapping objects, as discussed further below.

At block 188, a test is made to determine if the current object is to be trapped and, if so, the color and placement of the trap are set at block 190. The trap is then drawn according to the color and placement determinations, shown at block 192. Once the trap for the current object is drawn at block 192, or if it was determined that the current object was not to be trapped, or if the current object did not overlap or abut the trapped object as determined at block 182, a test is made at block 194 to determine whether any background objects that have yet to be considered in the shape directory. If there are objects remaining, the routine loops to block 180 where the next background object is retrieved from the shape directory. Otherwise, the routine terminates and control is returned to block 176 of FIGURE 7.

The decision of whether or not a trap should be created, as determined at block 188, may be based on a number of different criteria. For example, it may be acceptable to refrain from trapping text below a certain point size, e.g., 24 points. In a preferred embodiment, this is a user-definable variable. Step limits may also be implemented, such that a trap is not created if the relationship of the colors of two intersecting objects is within a given threshold. The threshold may be determined by realizing that the effort expended in trapping certain color intersections outweighs the benefit of the trap. It is noted that it may be more efficient to make at least some of these determinations earlier in the process, as opposed to making them at block 188. For additional criteria in determining whether or not a trap should be created, and for criteria on determining trap placement and color, please refer to U.S. Patent Application Serial No. 08/188,246, entitled "Applying Traps to a Printed Page Specified in a Page Description Language Format," which was already incorporated by reference.

FIGURE 9 is an exemplary subroutine that takes into account errors that may be created using the trapping program 20 when three or more objects overlap one another. A test is made at block 200 to determine whether the current object has been trapped before, i.e., has any trapped edges. If the current object has been trapped before, it is advantageous to re-

move the prior trap from any edge or portion thereof that intersects the background object. This is because the background object that is currently being considered may overwrite the former background object that caused the trap to be created in the first place. In turn, the former trap may interfere with a trap to be created between the current object and the background object currently being considered, producing spurious results. In other words, if it was determined that the current object has been trapped prior to this pass, and since it is known that the background object under consideration intersects the current object (from block 182 of FIGURE 8), it is preferable to eliminate the earlier trap at the areas in which there is an overlap. Please refer to FIGURES 14A-14C for additional explanation.

The elimination of an earlier-created trap, or portion thereof, may be accomplished by stroking the overlapping or affected edge(s) or portions thereof with the fill color of the background object, as shown in block 202. While this step will remove the previous trap from the background object, it will have the negative effect of drawing the background color onto part of the current object. This result is the phenomenon of the PS language, in that the entire centerline comprising the trap must be redrawn, and the redraw will extend into both the current and background objects.

To eliminate the negative effect of stroking the affected edge, the current object is filled with the current objects' fill color at block 204. Once the current object has been redrawn with its fill color, or if it was determined that there were no traps drawn against the current object at block 200, the subroutine terminates and control is returned to block 186 of FIGURE 8. Thereafter, a trap will be created, if desired, between the current and background objects at their intersecting edge(s).

The remainder of the Detailed Description and accompanying drawings set forth examples that facilitate a more thorough understanding of the invention. FIGURE 10 illustrates a simplified publication 210 having a prolog 212 and a script 214. The prolog 212 includes a single procedure 216 labeled "Rectangle" that may be used for creating rectangles and the like within the publication using the desktop publishing program. The script 214 includes two exemplary calls 218 and 220 to the procedure 216, each causing a rectangle to be drawn of an indicated size, at an appropriate position, and at a necessary rotation, as well as any other variables necessary for drawing the object. When the publication is to be printed, the prolog and script commands are sent to the RIP, which interprets, rasterizes and outputs the resultant bit map to a printer or film. It is noted that the commands shown in FIGURE 10 are in pseudo code, and do not necessarily correspond to actual PS commands.

FIGURE 11 illustrates the modification made to the prolog 212 of FIGURE 10 in order to accommo-

date trapping in accordance with the invention. Specifically, a trap prolog 222 to accomplish trapping includes the addition of two lines of pseudo code to the original prolog 212. Namely, line 224 stores objects in the shape directory as they are being drawn. As was discussed above, in a preferred embodiment, only the graphical objects are stored, and not text. The second line of code added, line 226, traps the object currently being drawn against all the other objects in the shape directory. Trapping of the object may be accomplished by the routines set forth in FIGURES 7 and 8 and accompanying text. It is noted that FIGURES 10 and 11 are illustrative only, and actual PS commands and subroutine must be inserted in order for the routines to be executable.

FIGURES 12A-12C illustrate a foreground Object Y that is to be trapped against a background Object X. This discussion begins with the routine of FIGURE 8, and assumes that both Object X and Y were drawn and added to the shape directory using the routine of FIGURE 7. At block 176, there is one additional object in the shape directory (Object X), and the trapping routine of FIGURE 8 is called. Object Y is the "current object" under consideration.

With reference to block 182 of FIGURE 8, since Object Y overlaps Object X (FIGURE 12B), it may need to be trapped. The clipping path is set at block 184 and Repair\_Trap is called at block 186. Since Object Y has not been trapped before (block 200, FIGURE 9), control returns to block 188 of FIGURE 8. Assuming that Object Y is to be trapped, the trap color and placement are determined at block 190 and a trap 230 is drawn at block 192. As there are no other objects in the shape directory, the process is complete and control returns to the routine of FIGURE 7, which has also been completed.

FIGURES 13A-13C illustrate the addition of a third Object Z to the above example, and further assumes that the routine Repair\_Trap is not implemented. Beginning with FIGURE 13A, the Object Z is drawn and added to the shape directory using blocks 172 and 174 of FIGURE 7. At block 176, there are additional objects in the shape directory (Objects X and Y), and the trapping routine of FIGURE 8 is called. Object Z is the "current object" under consideration.

With reference to block 182 of FIGURE 8, the background object first considered is Object X, as the objects from the shape directory are retrieved from the bottom up. Since Object Z overlaps Object X, it may need to be trapped against Object X. The clipping path is set at block 184. In this example, it is assumed for illustrative purposes that Repair\_Trap is not implemented, and thus block 188 is considered. Assuming that Object Z is to be trapped, the trap color and placement are determined at block 190 and a trap 232 is drawn at block 192. An exemplary trap 232 is shown in FIGURE 13B. The trap 232 is a trap resulting from the intersection of Objects X and Z, and Ob-

ject Y has not yet been taken into account. Once the trap for this intersection is drawn in block 192, a test is made at block 194 to determine if any objects remain in the shape directory that have not been considered. Object Y has yet to be considered, and the routine loops to block 180 where Object Y is retrieved. Object Y is now the "background object" being compared to the current object, Object Z.

With reference to block 182, since Object Z overlaps Object Y, it may need to be trapped against Object Y. The clipping path is set at block 184. Again, Repair\_Trap is not called due to the assumption stated above. Assuming that Object Z is to be trapped against Object Y, the trap color and placement are determined at block 190 and a trap 234 is drawn at block 192. Trap 234 is shown in FIGURE 13C. At this point, there are no other objects in the shape directory to be considered, and the process is complete. Thus, control returns to the routine of FIGURE 7, which has also been completed.

As can be seen, the result in FIGURE 13C is less than ideal, because the traps 232 and 234 have combined to form overlapping traps. FIGURES 14A-14C illustrate the effects of the Repair\_Trap routine to accommodate the problem that may occur when three or more objects overlap in the same area of a page, as shown in FIGURE 13A-13C. The explanation of FIGURES 13A-13C will be reexamined, with the supposition that Repair\_Trap is invoked.

With reference to block 182 of FIGURE 8, again the background object under consideration is Object X. Since Object Z overlaps Object X, it may need to be trapped against Object X. The clipping path is set at block 184 and Repair\_Trap is called at block 186. Since Object Z has not been trapped before (block 200, FIGURE 9), control returns to block 188 of FIGURE 8. Assuming, as above, that Object Z is to be trapped, the trap color and placement are determined at block 190 and the trap 232 drawn, as shown in FIGURE 13B. Once the trap 232 is drawn in block 192, a test is made at block 194 to determine if any objects remain in the shape directory that have not been considered. Object Y has yet to be considered, and the routine loops to block 180 where Object Y is retrieved. Object Y is now the "background object" being compared to the current object, Object Z.

With reference to block 182, since Object Z overlaps Object Y, it may need to be trapped against Object Y. The clipping path is set at block 184. FIGURE 14A and B indicate this step by showing Objects X and Z as dashed lines, e.g., these areas are not considered by the RIP in the reaming steps. Repair\_Trap is again called at block 184. With reference to FIGURE 9, the current object, Object Z, has been previously trapped (trap 232), and the routine continues to block 202. At block 202, the edges at which Objects Z and Y overlap are stroked using the fill color of Object Y. This is indicated in FIGURE 14A, with the dash-

ed area 238. As can be seen, this step has the positive effect of removing the trap 232 from the effected area of Object Y, designated by reference numeral 238a. However, the step has the negative effect of overwriting a portion, area 238b, of Object Z. Thus, at block 204, Object Z is filled with its own fill color, shown by area 240. As will be appreciated, filling Object Z with its own color will erase any portion of the trap 232 that might remain in area 240, i.e., that may have extended beyond the area 238b. Having completed block 204, the routine returns to block 188 of FIGURE 8.

At block 188, a test is made to determine if Object Z should be trapped against Object Y. Assuming, as above, that Object Z is to be trapped against Object Y, the trap color and placement are determined at block 190 and the trap 232 is drawn at block 192. The trap 234 is again created, as shown in FIGURE 14C. It is noted that as a result of the Repair\_Trap procedure, the trap 232 no longer includes the right hand corner of Object Z, which is also shown in FIGURE 14C. After drawing the trap 234 at block 192, a test is made at block 194 to determine if there are any shapes in the shape directory that have not been considered. At this point, there are no other objects in the shape directory to be considered, and the process is complete. Thus, control returns to the routine of FIGURE 7, which has also been completed.

As will be appreciated, the resultant traps in FIGURE 14C are superior to the solution shown in FIGURE 13C because the effects of earlier traps have been removed. It will also be appreciated that other methods may be used to achieve this same result, and the invention is not to be limited to the particular routine set forth in FIGURE 9.

As will be appreciated from the foregoing, a trapping program in accordance with the teaching of the invention allows trapping capabilities to be provided to originating programs without the need to change the code in the originating programs. The trapping program thus runs on top of and completely transparent to the originating program.

While the preferred embodiment of the invention has been illustrated and described, it will be appreciated that various changes can be made therein without departing from the spirit and scope of the invention. For example, the instructions to the RIP to trap PDL files may be sent to the RIP independently, or be resident at the RIP, separate from any particular PDL file.

50

#### Claims

- 55 1. A method of electronically trapping a printed page including a plurality of objects expressed in a page description language, the page description language instructions being rendered by a raster image processor for output to an output de-

vice, the method comprising the step of providing instructions to the raster image processor to trap the objects in the page as the page description language instructions comprising the objects are interpreted by the raster image processor.

5

2. The method of Claim 1, wherein the printed page is in a publication in a desktop publishing program, the publication including a prolog and a script that specify the objects to be printed, the method further including the step of modifying the publication prolog to provide the instructions to trap to the raster image processor.

10

3. The method of Claim 2, wherein the script is sent unmodified to the raster image processor.

15

4. The method of Claim 1, wherein the page description language instructions comprising the objects are sent unmodified to the raster image processor.

20

5. A method of electronically trapping a printed page within a publication in a desktop publishing program, the publication including a prolog and script that specify a plurality of color objects when being rendered by a raster image processor, the method comprising the steps of:

25

(a) modifying the publication prolog to instruct the raster image processor to (i) create a directory of the color objects in the publication and (ii) trap the color objects in the publication using the directory; and

30

(b) sending the modified publication prolog and script to the raster image processor.

35

6. The method of Claim 5, wherein the areas are created as the publication is interpreted.

40

45

50

55

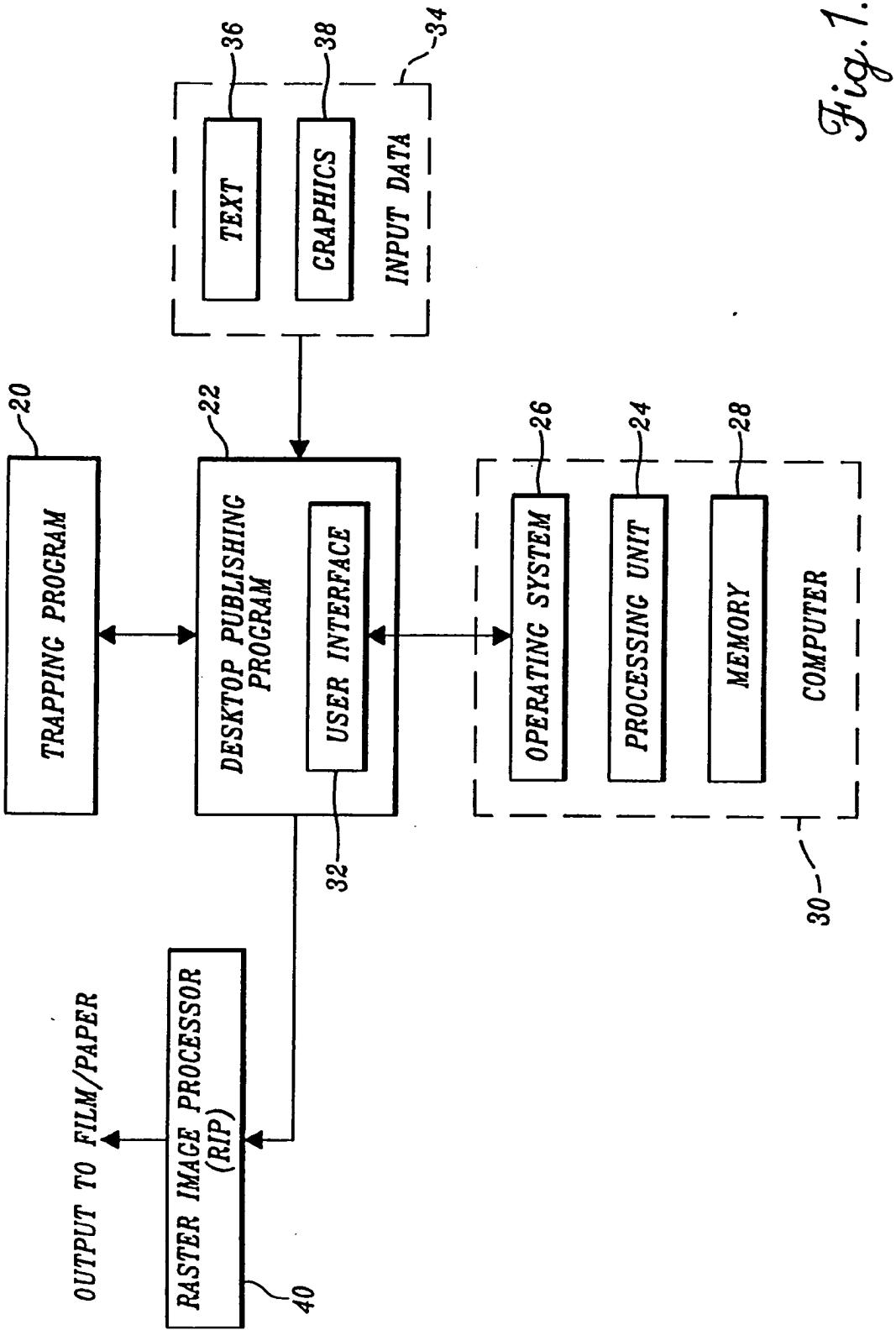
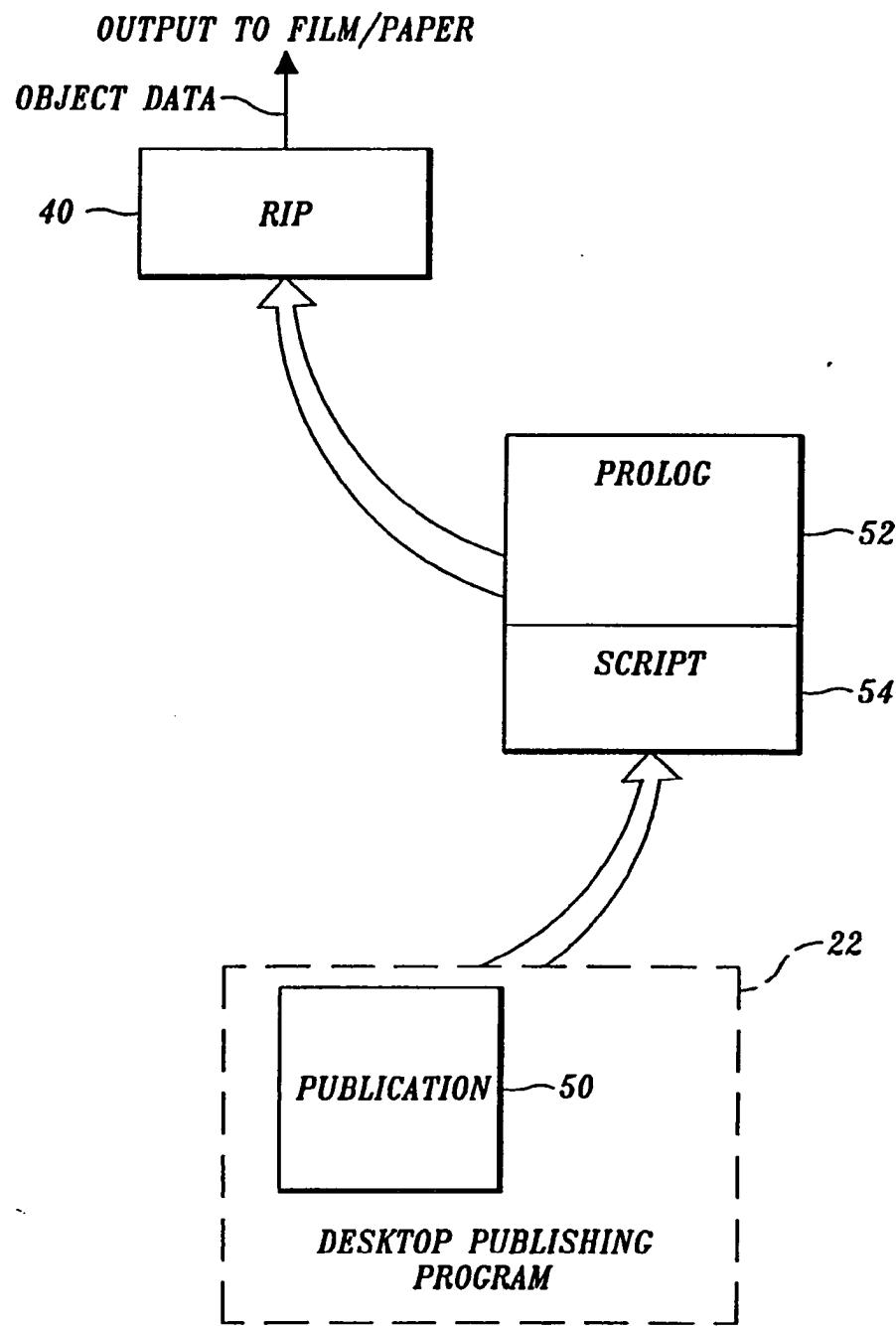
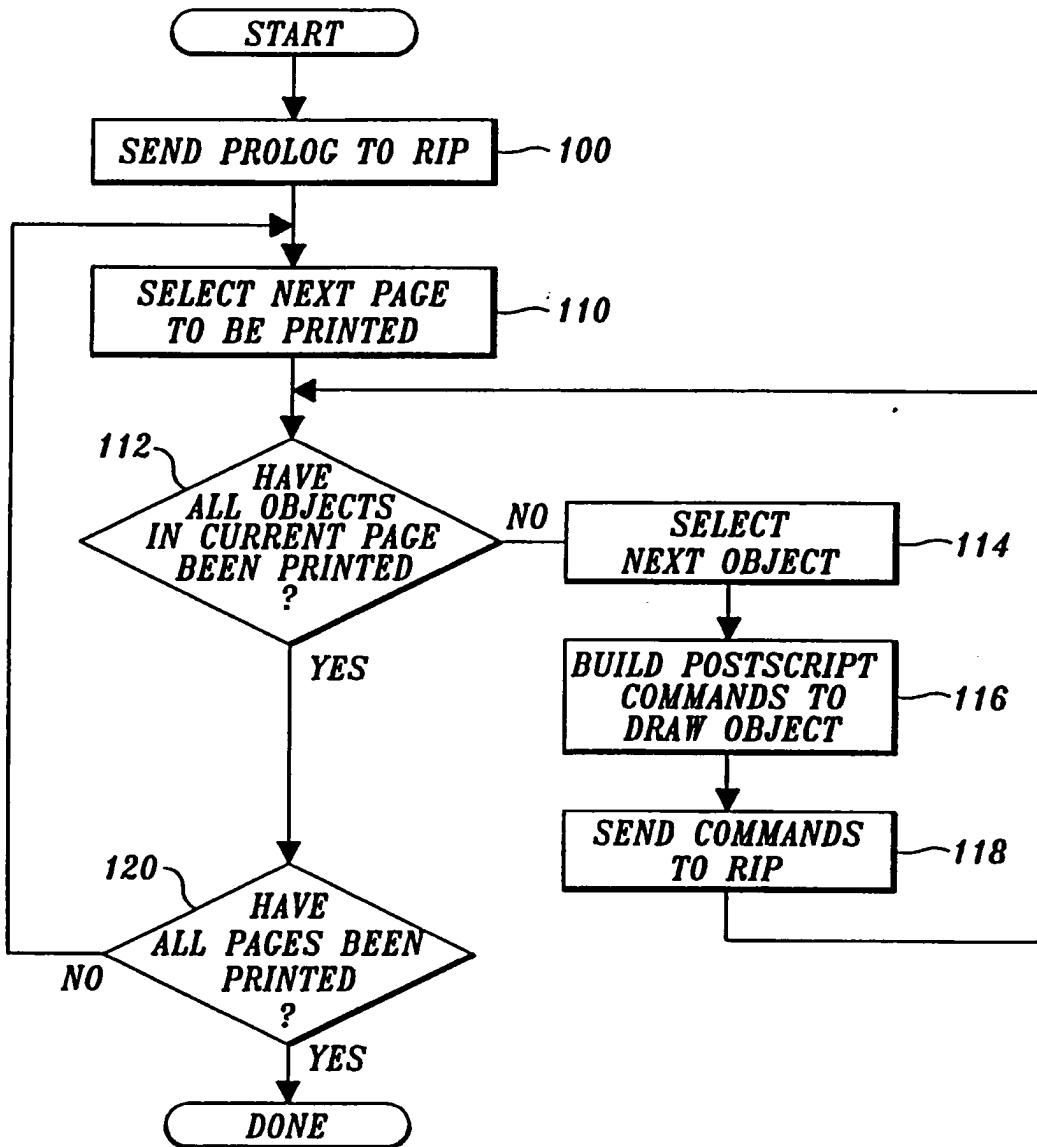


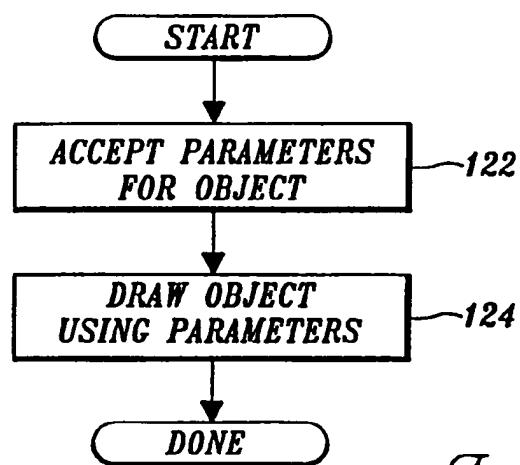
Fig. 1.



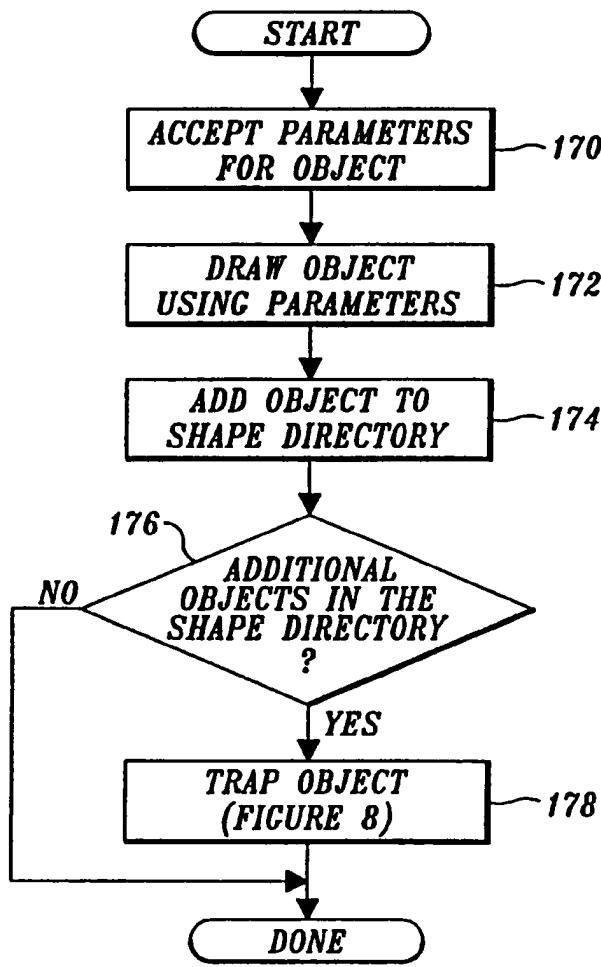
*Fig.2.  
PRIOR ART*



*Fig. 3.*  
PRIOR ART



*Fig.4.*  
PRIOR ART



*Fig. 7.*

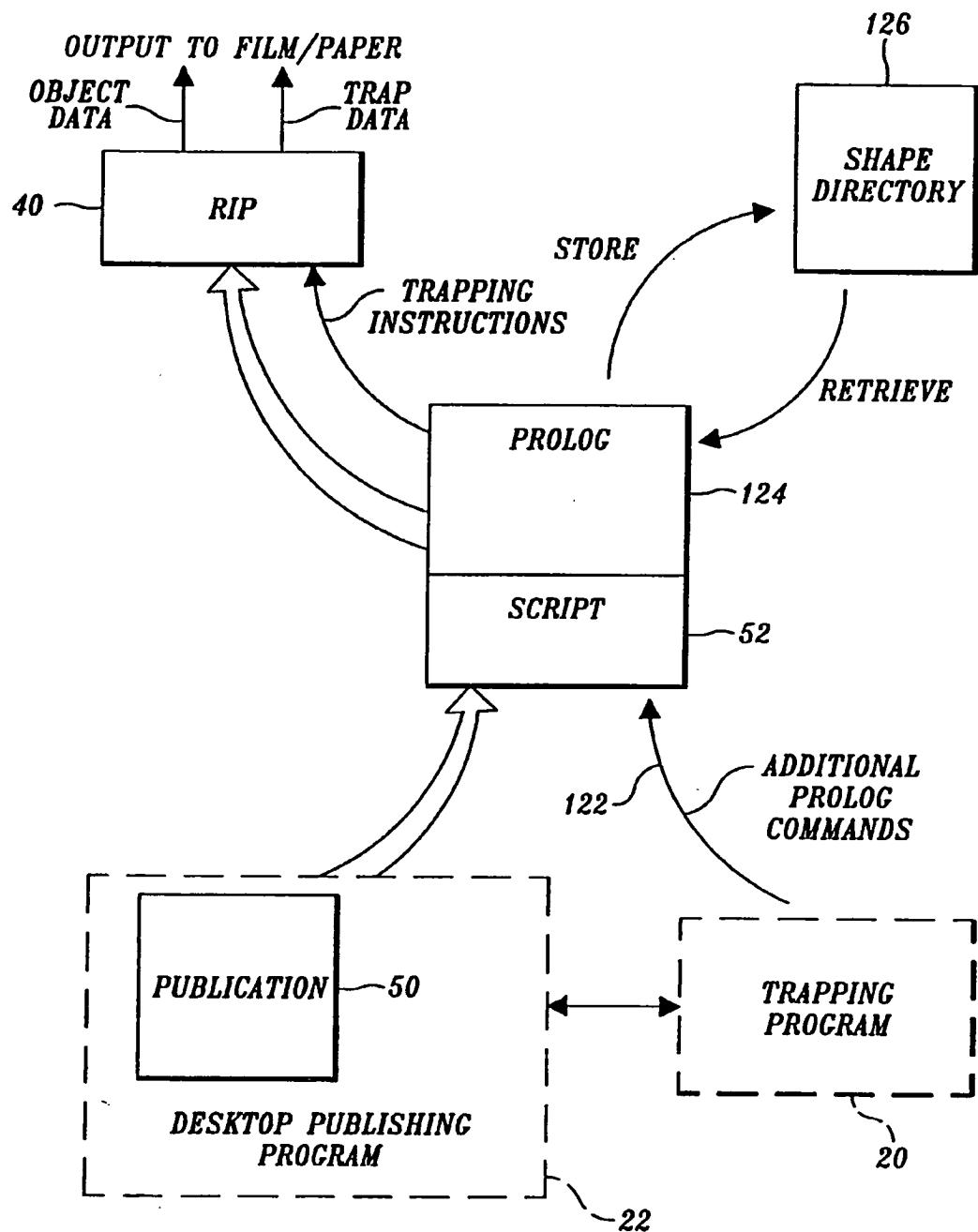


Fig.5.

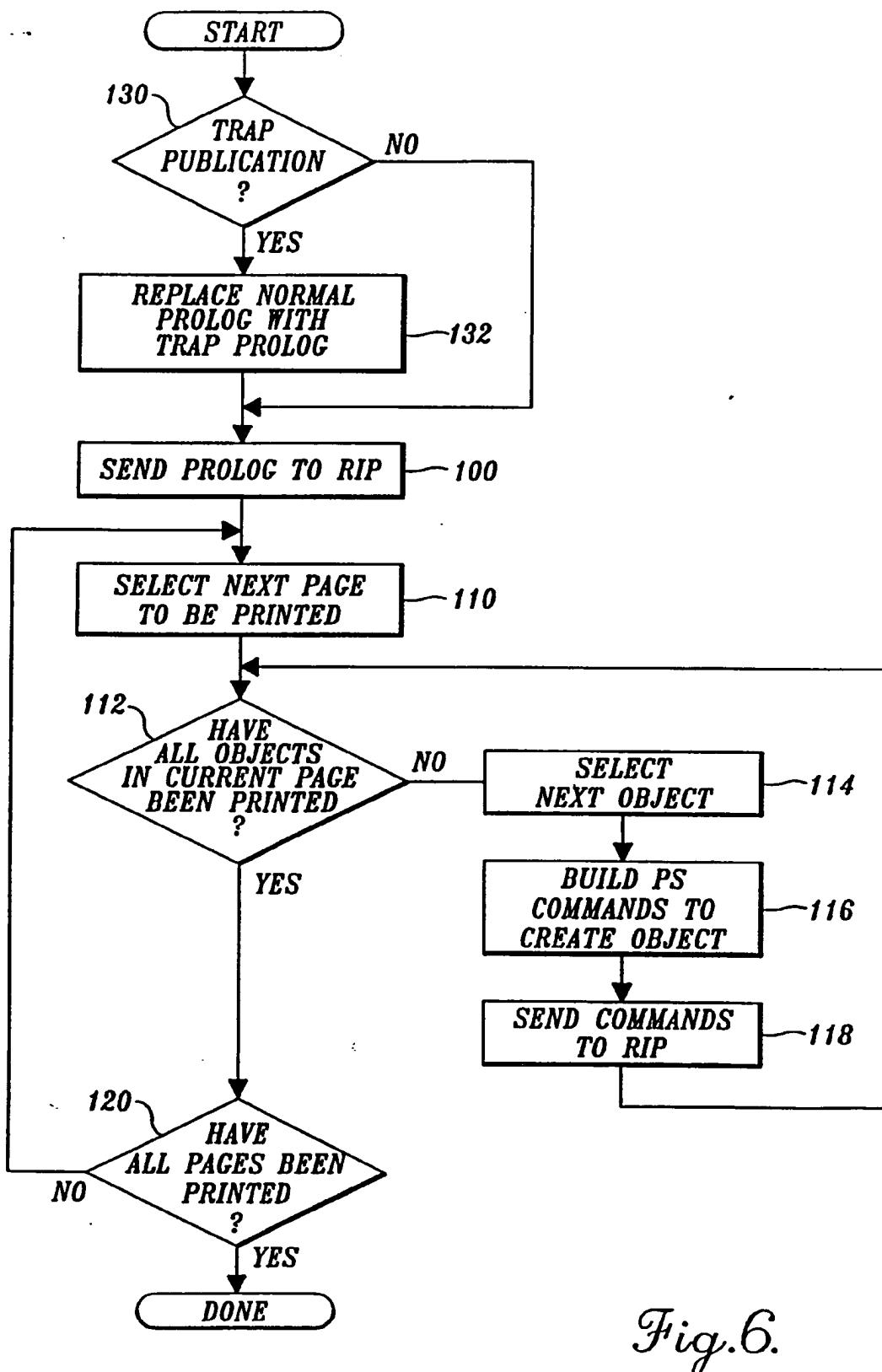


Fig.6.

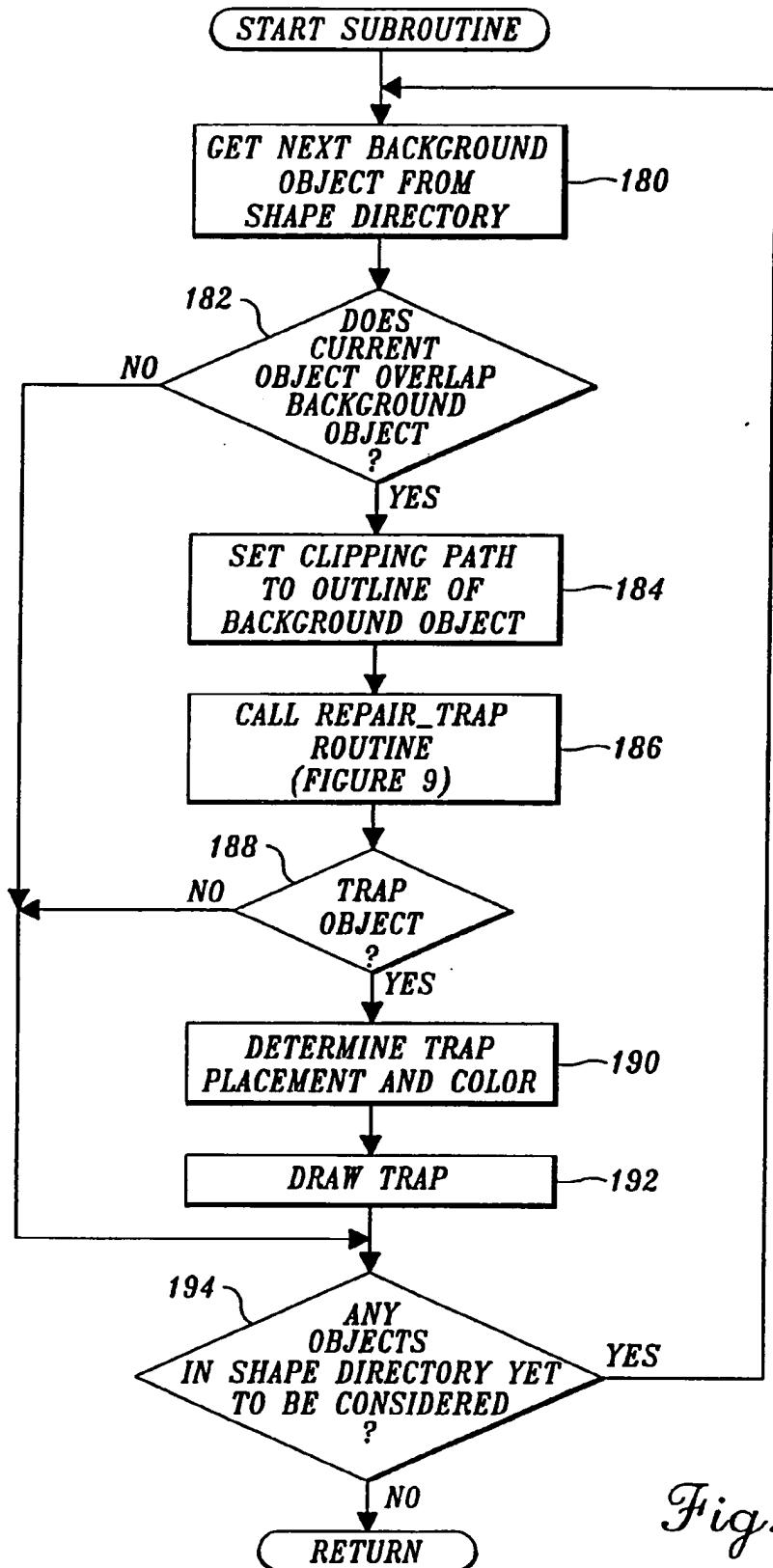


Fig. 8.

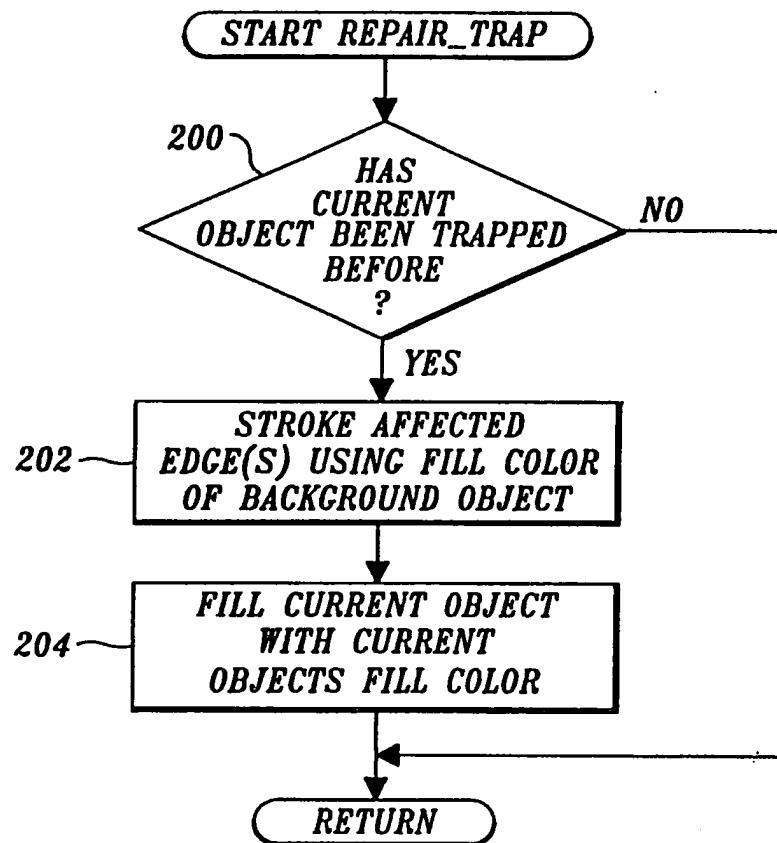


Fig. 9.

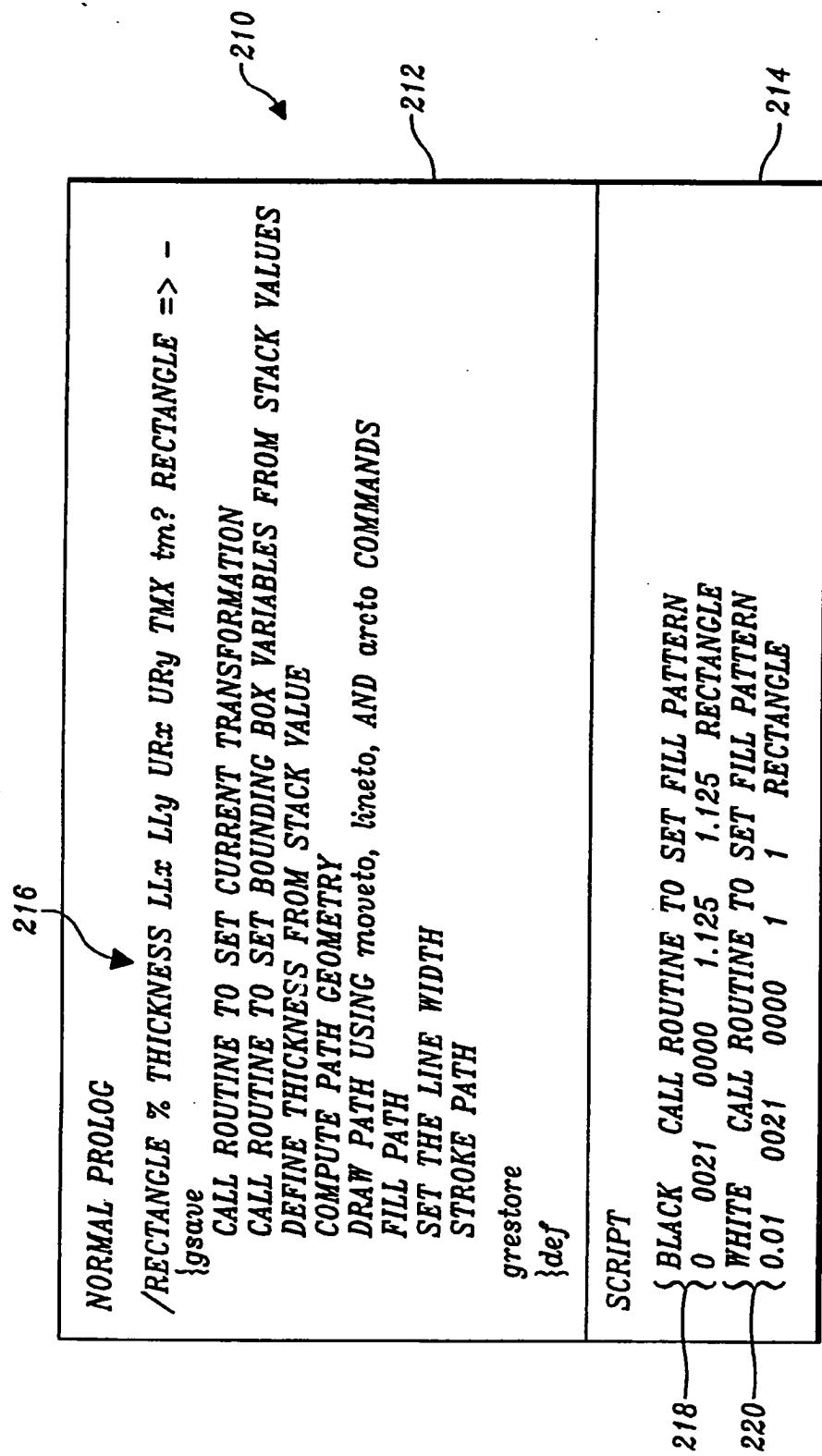


Fig. 10.

```

216
TRAP PROLOG
    /RECTANGLE % THICKNESS LLx LLy URx URy TMX tm? RECTANGLE => -
    {gsave
        CALL ROUTINE TO SET CURRENT TRANSFORMATION
        CALL ROUTINE TO SET BOUNDING BOX VARIABLES FROM STACK VALUES
        DEFINE THICKNESS FROM STACK VALUE
        COMPUTE PATH GEOMETRY
        STORE OBJECT IN SHAPE DIRECTORY
        DRAW PATH USING moveto, lineto, AND arcto COMMANDS
        FILL PATH
        SET THE LINE WIDTH
        STROKE PATH
224
    } -TRAP THE OBJECT AGAINST ALL OTHER OBJECTS IN SHAPE DIRECTORY
    grestore
    {def
    }

226
SCRIPT
    {BLACK CALL ROUTINE TO SET FILL PATTERN
    0 0021 0000 1.125 1.125 RECTANGLE
    {WHITE CALL ROUTINE TO SET FILL PATTERN
    0.01 0021 0000 1 1 RECTANGLE
220
    } 214

```

Fig. 11.

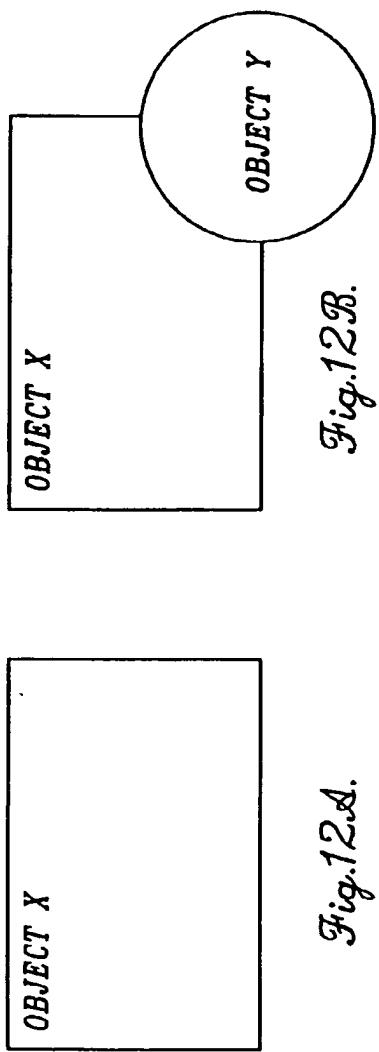


Fig.12B.

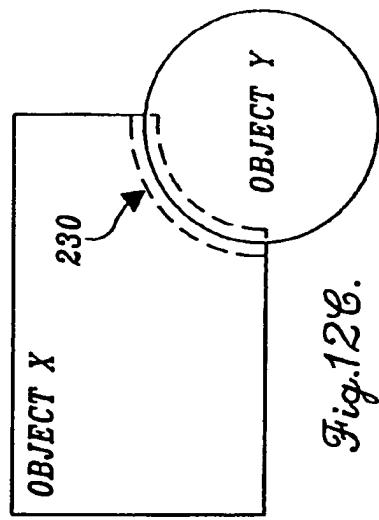


Fig.12C.

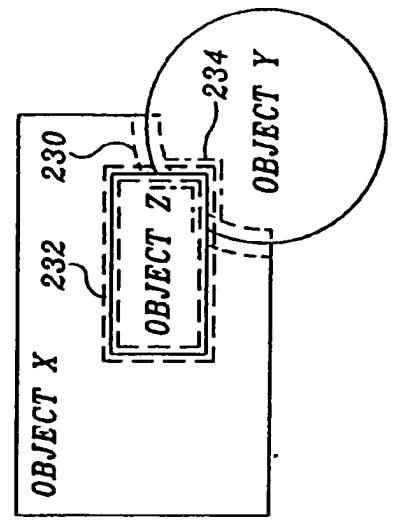


Fig.13C.

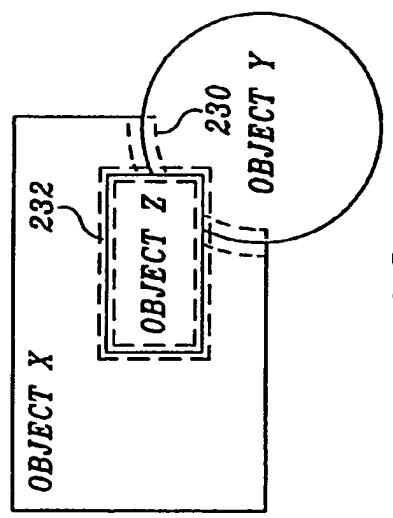


Fig.13B.

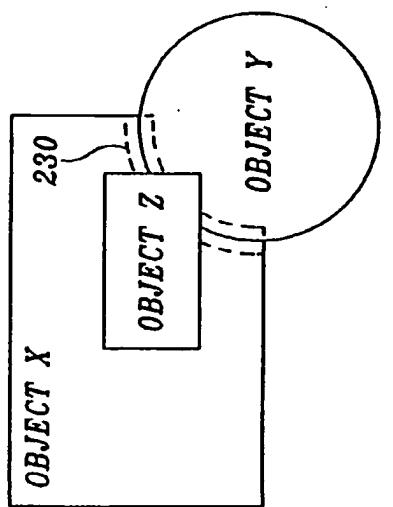


Fig.13A.

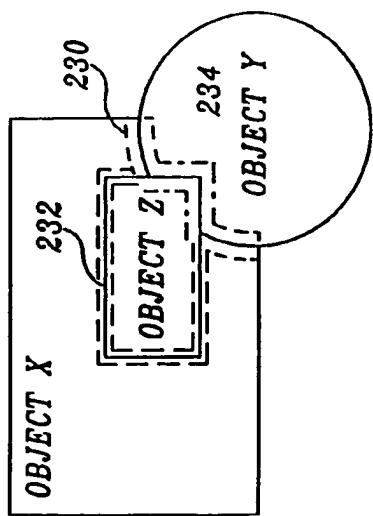


Fig.14C.

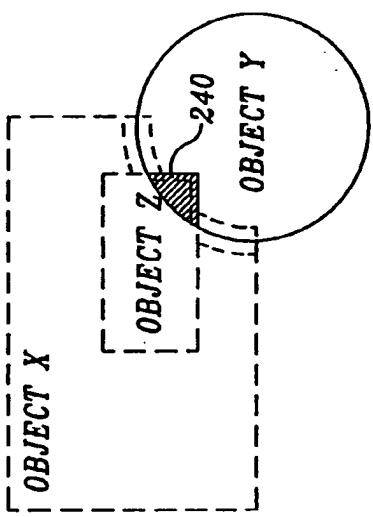


Fig.14B.

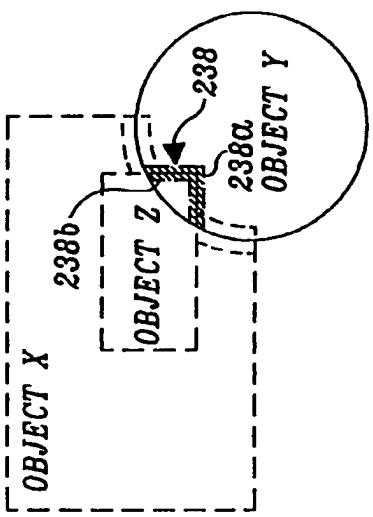


Fig.14A.